ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

CAMPUS DI CESENA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in
Sistemi Distribuiti

STRUMENTI PER MIDDLEWARE DI COORDINAZIONE: IL CASO DI TUCSON

CANDIDATO:
Andrea D'Intino

RELATORE:
Prof. Andrea Omicini

CORRELATORE:
Dott. Stefano Mariani

Anno Accademico 2015/16

Sessione II

Indice

Introduzione

L'importanza di avere un set completo di tool quando si lavora su una piattaforma informatica è ben nota a chiunque sia impiegato in ambito di programmazione e progettazione di sistemi software, o a chiunque abbia avuto voglia per diletto di cimentarsi in qualunque prova personale. Ad ognuno di noi è capitato sicuramente almeno una volta di rimanere bloccati durante la stesura di un codice o di non avere ben chiaro come mai il risultato dell'esecuzione del programma appena creato differisse da quanto ci aspettassimo. Ecco, in questi casi è molto probabile che sia stato un qualche tool a nostra disposizione a venirci in aiuto per toglierci dai pasticci. Questo è il motivo per il quale per set di tool in informatica si intende un insieme di strumenti software di base per facilitare e uniformare lo sviluppo di applicazioni derivate più complesse o, più in generale, qualsiasi strumento che ci fornisce il sistema per monitorare nel dettaglio ciò che sta accadendo.

Durante lo sviluppo di questa tesi andremo a studiare e lavorare in ambiente TuCSoN, il nostro scopo sarà dunque quello di arricchire il set di strumenti che attualmente ci vengono forniti dal sistema con un nuovo tool che si è dimostrato poter essere interessante.

Nel primo capitolo verrà data al lettore una panoramica generale su cosa siano i sistemi distribuiti, l'evoluzione dei sistemi dagli albori dell'informatica ad oggi. Verrà introdotto il concetto di middleware, spiegato a cosa serve e

analizzate le caratteristiche principali. Toccheremo inoltre problematiche importanti quali la comunicazione e la coordinazione di programmi distribuiti sulla rete fornendo alcuni cenni sui modelli più famosi.

Arriveremo poi ad analizzare cos'è il modello di coordinazione TuCSoN, quali tool ci offre il sistema e cosa questi ci permettono di fare.

Infine, nel terzo capitolo, entreremo nel cuore della nostra ricerca, ovvero partiremo da un problema (più precisamente la necessità di costruire uno strumento nuovo che possa tornare utile in futuro) lo analizzeremo e spiegheremo nel dettaglio la soluzione adottata e la successiva costruzione dell'applicazione ideata.

1 Background

In questo capitolo verrà introdotto tutto ciò che c'è da sapere per poter poi capire con chiarezza il lavoro che andremo a fare.

Partiamo dal principio, all'inizio della storia i sistemi informatici erano composti da grandi calcolatori con un'enorme capacità computazionale. A quei tempi dunque il problema principale di chi progettava software era la ricerca del miglior algoritmo per risolvere il problema che ci si trovava davanti.

Oggi le cose sono molto cambiate, i sistemi computazionali sono diventati pervasivi, ovunque ci voltiamo possiamo notare ed utilizzare strumenti che al proprio interno hanno dei calcolatori. Siamo quindi passati da enormi terminali la cui principale mansione era la pura elaborazione e calcolo algoritmico ad un'infinità di piccoli terminali che svolgono le più svariate funzioni. La maggior parte dei sistemi con cui veniamo a contatto oggigiorno hanno un cuore software e allo stesso tempo una natura sostanzialmente fisica e soprattutto distribuita. Ed è proprio in questo momento che inizia a cambiare la visione di chi progetta software, perché non deve più preoccuparsi solamente di trovare un giusto algoritmo per eseguire a dovere il proprio compito, bensì deve anche gestire questa distribuzione fisica del nuovo sistema che si viene a creare, il quale non è più composto solamente dal proprio "orticello" ma anche dall'infinità di altri terminali con cui si verrà a contatto. Tutto questo porta con se un altro concetto fondamentale,

quello del tempo. Già, perché se prima il sistema completo era chiaramente sotto il nostro controllo e potevamo immaginare una serie di operazioni a cui dare un ordine preciso, tutto questo ora non è più possibile. Non posso sapere cosa stia facendo in questo momento un'altra macchina con la quale ho bisogno di comunicare, e soprattutto non esiste più una nozione di tempo comune. Nasce quindi la necessità di costruire modelli che permettano la coordinazione e la comunicazione tra processi che potenzialmente si trovino alle estremità opposte della terra.

1.1 Sistemi Distribuiti e Middleware

Questi nuovi sistemi prendono il nome di sistemi distribuiti. Un sistema distribuito è una collezione di computer indipendenti che agli utenti appaiono come un sistema unico coerente [TVS07]. In questa definizione non c'è nessun tipo di traccia che faccia riferimento alla natura, alla struttura, al comportamento dei singoli sistemi. Sappiamo che parlando di sistemi distribuiti sbatteremo contro la nozione di eterogeneità. Qui nasce quindi il primo punto fondamentale dei sistemi distribuiti; la collaborazione: molte entità autonome che devono lavorare insieme come un singolo sistema coerente.

Queste entità autonome devono lavorare insieme collaborando, devono amalgamarsi pur essendo eterogenee. Al fine di supportare computer e reti anch'esse eterogenee, continuando però a garantire la visione di un singolo sistema, i sistemi distribuiti sono spesso organizzati come uno strato (layer) di software logicamente posizionato tra uno strato di livello superiore, costituito dagli utenti e dalle applicazioni, e uno strato di livello inferiore costituito dal sistema operativo e dalle funzionalità di comunicazione di base. Tali sistemi distribuiti sono anche chiamati middleware.

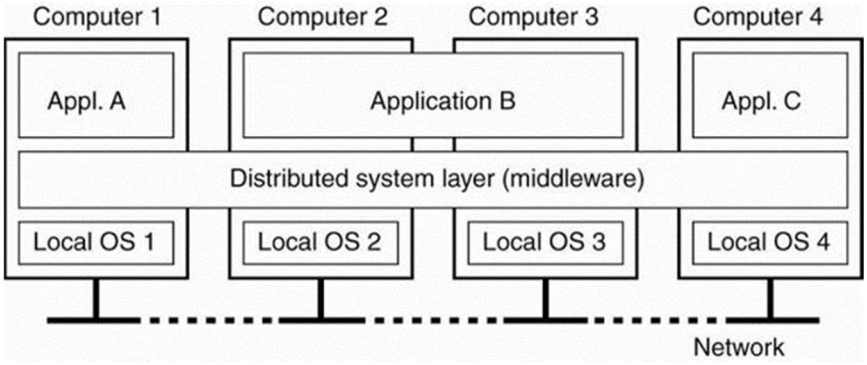

Figura 1 – Distribuited System Layer (middleware) [TVS07]

La figura 1 mostra quattro computer in rete e tre applicazioni, di cui l'applicazione B è distribuita sui computer 2 e 3. Ad ogni applicazione è fornita la stessa interfaccia. Il middleware fornisce ai componenti di una singola applicazione distribuita il mezzo per comunicare tra loro ed allo stesso tempo nasconde ad ognuna di loro le differenze di hardware e sistema operativo. Le applicazioni che insistono sui singoli computer sono nella parte alta; la visione architetturale classica è inserire un layer middleware che fa da tramite, si occupa cioè di rimanere esteso sopra tante macchine e offrire a tutte la stessa interfaccia così da poter vedere la molteplicità di sistemi come un unico grande sistema. Stiamo dunque passando ad un concetto di sistema distribuito immettendo un layer middleware che introduce una soluzione nascondendo le differenze tra le parti.

Come detto in precedenza, uno degli aspetti chiave di questi nuovi sistemi distribuiti è la comunicazione. C'è bisogno di un modello che permetta di comunicare alle diverse parti che compongono il nostro sistema. La comunicazione può essere sincrona o asincrona:

- Comunicazione Asincrona: Il mittente, subito dopo aver inviato il proprio messaggio, continua l'elaborazione. Questo può avvenire perché è il middleware che si occupa di memorizzare temporaneamente il messaggio.

- Comunicazione Sincrona: Il mittente è bloccato finché la sua richiesta non viene accettata. Ci sono tre punti nei quali la sincronizzazione può aver luogo; Il mittente può essere bloccato finché il middleware non comunica che prenderà il controllo della trasmissione della richiesta, oppure, il mittente si può sincronizzare quando la richiesta è stata consegnata al destinatario indicato, o ancora, la sincronizzazione può avvenire lasciando che il mittente attenda finché la sua richiesta non sia stata completamente elaborata.

1.2 Coordinazione

Andiamo ora ad analizzare l'ultimo punto fondamentale di cui andremo ad occuparci, e cioè la coordinazione. Il modello che interessa a noi è quello basato sulle tuple.

Definiamo quindi innanzitutto cos'è un tupla: una tupla è una collezione ordinata, anche eterogenea, di elementi conosciuti con una struttura a record. Contiene tutte le informazioni di un certo tipo (aggregazione semantica). Le tuple non hanno la necessità di avere nomi dei campi, permettono una facile aggregazione di conoscenza e hanno una semantica di interpretazione, cioè ogni tupla contiene tutte le informazioni che riguardano un tema dato. La struttura delle tuple è basata su: arità (cioè il numero degli elementi della tupla), tipo, posizione e contenuto informativo. Le anti-tuple o tuple template invece si utilizzano per descrivere set di tuple.

Per la coordinazione Tuple-based, si può definire un Meta-Modello [PC96] la cui base concettuale è quella di avere delle entità coordinate e coordinabili che cooperano e si sincronizzano tramite tuple, le quali sono disponibili nello spazio di tuple, attraverso accesso associativo consumandone di esistenti e producendone di nuove. Ogni componente può scrivere

la propria tupla, proprio perchè si sta tentando di realizzare sistemi coordinati e interattivi.

- Il medium di coordinazione è il cosiddetto “spazio delle tuple”. All’interno degli spazi di tuple possiamo ritrovare anche tuple identiche tra loro.
- Il linguaggio di comunicazione è costituito dalle tuple stesse. Le tuple sono un modo per scambiarsi informazioni e si possono definire come collezioni ordinate di elementi informativi potenzialmente eterogenei.
- Il linguaggio di coordinazione è formato dalle primitive dello spazio delle tuple, che è una collezione di operazioni che consentono di scrivere, leggere e consumare tuple

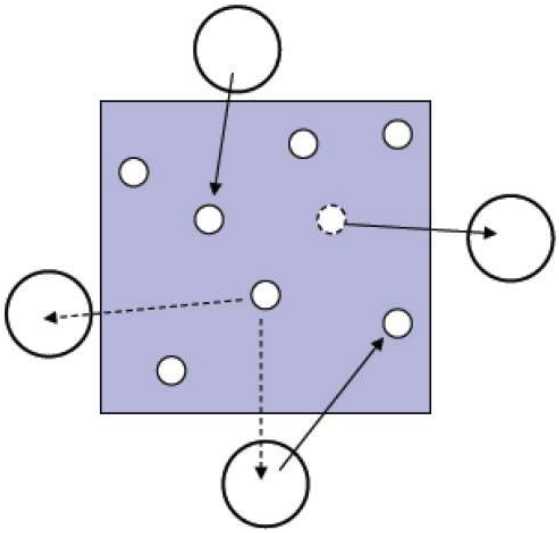

Figura 2 – Diversi processi possono scrivere, leggere o consumare tuple nello spazio comune

Negli anni ’80, nell’ambito del calcolo parallelo, nasce Il modello di coordinazione cosiddetto LINDA. Per quanto riguarda il linguaggio di comunicazione, abbiamo già definito le tuple come una collezione ordinata di possibili informazioni eterogenee. In questo ambito, ci deve essere un qualcosa

atto a specificare una tupla di un certo tipo; a tal fine si usano i tuple template detti anche anti-tuple che sono specificazioni di set/classi di tuple. È necessario anche un meccanismo di matching che servirà per fare corrispondere tuple e template. Questa associazione tuple-template che serve per selezionare un certo tipo di tuple viene chiamata Accesso Associativo. Per quanto riguarda il linguaggio di coordinazione, possiamo definire alcune primitive fondamentali:

- out(T): Inserisce la tupla T nello spazio di tuple.
- in(TT): Recupera, per poi cancellarlo, il tuple-template TT dallo spazio di tuple, con la seguente semantica:
    - Se ce n'è solo una, viene presa, restituita al chiamate e poi eliminata: Lettura distruttiva (Desctructive Reading);
    - Se ce ne sono più d'una, ne viene scelta una in modo non deterministico: Non-Determinismo (Non-Determinism);
    - Se non si ha la corrispondenza tupla-template nello spazio di tuple, l'operazione di esecuzione viene sospesa e ripresa non appena la tupla giusta viene trovata: Semantica Sospensiva (Suspensive Semantics).

Con la Semantica Sospensiva, due sistemi riescono a coordinarsi sospendendo l'esecuzione e aspettando che l'altro abbia finito di fare quello che deve fare, dopodiché si riprende la comunicazione nello stesso momento (quindi sincronizzati l'uno con l'altro). Per quanto riguarda le primitive predicative possiamo definire inp(TT) e rdp(TT). Entrambe recuperano il tuple-template TT dallo spazio di tuple. Se viene trovata una tupla corrispondente essa viene restituita altrimenti viene mostrato un messaggio di avviso (fallimento). Le primitive, incluse quelle predicative, trattano una tupla per volta; alcuni problemi di coordinazione richiedono di gestire più di una tupla alla volta tramite una singola primitiva. In questi casi

intervengono rd_all e in_all che prendono tutte le tuple nello spazio delle tuple che matchano con TT e le restituiscono; in sostanza fanno ciò che fanno le query. Queste due primitive dette bulk non hanno una semantica sospensiva, quindi se nessuna tupla che soddisfa i requisiti viene trovata, viene restituita una collezione vuota. Non hanno nemmeno una semantica di successo/fallimento poiché una tupla viene sempre restituita, al massimo, come già detto, viene restituita una tupla vuota. Infine hanno una versione anche non distruttiva, dato che, mentre in_all(TT) distrugge tutte le tuple presenti nello spazio di tuple che soddisfano le richieste, rd_all(TT) lascia lo spazio di tuple intonso.

Lo spazio di tuple Linda può essere visto come un collo di bottiglia per la coordinazione, infatti se in un sistema semplice c'è un problema di coordinazione, nei sistemi più complessi ci sono n problemi di coordinazione che vanno risolti contemporaneamente; per questo motivo si pensò di creare più spazi di tuple disponibili per i processi. Si pensò cioè di divedere lo spazio di tuple in tanti spazi di cui ognuno incapsula una porzione del carico della coordinazione. In sostanza si dividono e si incapsulano diversi tipi di attività in spazi di tuple separati. È necessaria una sintassi che sarà dipendete dal modello e dall'implementazione. Un esempio di operazione può essere la seguente: *ts@node ? out(p)* che fa out(p) nello spazio ts al nodo node.

Le proprietà fondamentali di questo modello sono:

- Comunicazione generativa: fino al momento di un prelievo, le tuple generate dai coordinabili hanno un'esistenza indipendente all'interno dello spazio delle tuple; una tupla è equamente accessibile da tutti i coordinabili ma non è collegata a nessuno di questi. L'ortogonalità della comunicazione permette a mittenti e destinatari di inviarsi informazioni senza bisogno di conoscersi in quanto sono disaccoppiati nello spazio (non è necessario che due processi coesistano nello spazio per interagire), nel tempo (non è necessaria

la simultaneità per interagire) e nel nome (non è necessario avere dei nomi affinché i processi possano interagire tra di loro).

- Accesso associativo: le tuple nello spazio di tuple sono accedute attraverso il loro contenuto e la loro struttura ma soprattutto attraverso il nome, l'indirizzo o la locazione. Nella comunicazione content-based la sincronizzazione è basata sul contenuto e sulla struttura delle tuple. La reifica consiste nel trasformare l'evento astratto in una tupla concreta, raggruppando le classi di eventi in tuple e accedendo a queste tramite il template.

- Semantica sospensiva: le operazioni possono essere sospese per la non reperibilità di tuple che soddisfino i requisiti e riprese nel momento in cui queste tuple diventino disponibili. La semantica sospensiva ha l'idea che quando arriva una primitiva sospensiva e la tupla non è disponibile, si sospende il tutto e si riparte quando diventa disponibile la tupla.

Introduciamo l'ultimo concetto, cioè quello di centri di tuple ReSpecT: essi adottano le tuple logiche sia per le tuple ordinarie che per le tuple specifiche. Le tuple ordinarie sono fatti logici di prim'ordine, mentre le tuple specifiche sono tuple logiche con questa forma: reaction(E, G, R) dove E→evento, G→guardia, R→reazione. Se si verifica un evento e la guardia è verificata allora avviene una reazione. I centri delle tuple ReSpect incapsulano la conoscenza in termini di tuple logiche e il comportamento in termini di specifiche ReSpect. I centri di tupla ReSpecT sono:

- Ispezionabili: durante il funzionamento delle entità computazionali del sistema si può non solo osservare il comportamento del sistema ma anche lo stato del comportamento delle entità da cui dipende.

- Malleabili o forgiabili, quindi modificabili a run-time e non solo a design time.

- Situated: è il fatto di essere immersi in un ambiente e di comportarsi di conseguenza [CO09]. Si può essere situati nel tempo oppure nel mondo esterno. La cosa fondamentale è avere il corretto modello degli eventi.

La situatedness ha una stretta correlazione con l'ambiente e l'interazione. Tecnicamente è l'abilità di percepire e reagire ai cambiamenti nell'ambiente, riguarda quindi l'interazione tra processi e ambiente, per questo la si può pensare come un problema di coordinazione nel quale ci si chiede come trattare e governare le interazioni tra i processi proattivi e l'ambiente in continuo divenire.

I sistemi distribuiti sono immersi nell'ambiente e devono essere reattivi ad ogni sorta di evento, perciò il media di coordinazione deve avere la capacità di mediare qualsiasi attività nei confronti dell'ambiente per poter garantire corrette interazioni.

1.3 TuCSoN: Modello e Strumenti

Finita questa panoramica più generale sui sistemi distribuiti e sui modelli più famosi, entriamo ora più nel dettaglio iniziando a parlare di quello che sarà il vero e proprio protagonista di questa tesi, ovvero il middleware TuCSoN.

TuCSoN è l'acronimo di Tuple Centres Spread over the Network ed è un modello per la coordinazione di processi distribuiti, nonché di agenti autonomi [OZ99]. Le entità di base di questo modello sono:

- Agenti TuCSoN, ovvero i coordinabili.
- I Tuple Centres ReSpecT, cioè il media di coordinazione [OD01]

- I nodi TuCSoN, che rappresentano le entità di base che ospitano i tuple centres

Ognuno di questi tre elementi ha un identificatore univoco all'interno del sistema. Genericamente parlando, un sistema TuCSoN è una collezione di agenti e tuple centres che lavorano insieme in un sistema di nodi distribuiti. Dal momento che gli agenti sono soggetti proattivi, mentre i centri delle tuple sono solitamente reattivi, c'è bisogno di un insieme di operazioni di coordinamento per agire sul media di coordinazione. Queste operazioni sono fornite dal TuCSoN coordination language, definito da una collezione di TuCSoN coordination primitives che gli agenti possono utilizzare per scambiarsi le tuple. I centri delle tuple invece forniscono lo spazio condiviso per la comunicazione basata sulle tuple (tuple space), insieme allo spazio programmabile dei comportamenti per la coordinazione basata sulle tuple (specification space).

Qualunque nodo all'interno di un sistema TuCSoN è univocamente identificato da una coppia < NetworkId, PortNo >, dove:

- Il NetworkId è l'indirizzo IP del dispositivo che ospita il nodo.
- Il PortNo è il numero di porta su cui il TuCSoN coordination service rimane in ascolto in attesa delle richieste in entrata. La porta di default è la 20504.

Dato qualsiasi dispositivo collegato in rete con una Java Virtual Machine, un nodo TuCSoN può essere avviato tramite il comando:

java -cp tucson.jar;2p.jar alice.tucson.service.TucsonNodeService

Dove nei sistemi apple il ";" va sostituito con ":"

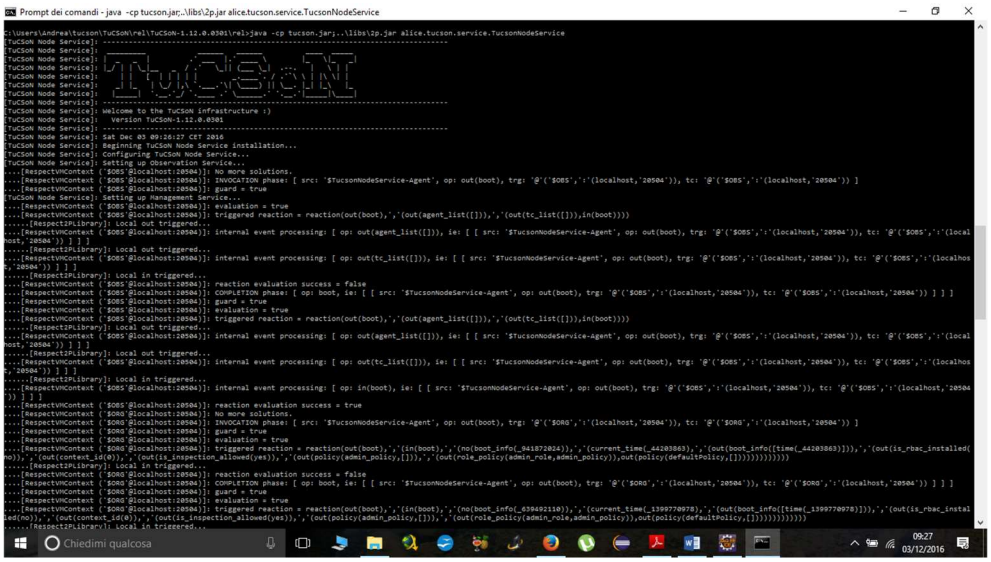

Figura 3 – Avvio di TuCSoN da linea di comando

Qualsiasi termine prolog-like di prim'ordine non contenente variabili è un nome ammissibile per il tuple centre [Llo84]. Qualunque centro delle tuple è univocamente identificato dal proprio nome e dall'identificatore del nodo nel quale risiede. Quindi il nome completo di un tuple centre ha la forma

tname @ *NetworkId* : *PortNo* . Il tname di default è, appunto, *default*.

Il linguaggio di coordinazione TuCSoN permette agli agenti di interagire con i centri delle tuple eseguendo operazioni di coordinazione. Il sistema fornisce primitive di coordinamento permettendo agli agenti di leggere, scrivere e consumare tuple. Qualsiasi operazione di coordinazione è invocata da un agente su un determinato centro delle tuple che si prende carica della sua esecuzione. Ogni operazione ha due fasi:

- Invocazione: richiesta da parte dell'agente nei confronti del tuple centre, comprensiva di ogni informazione.
- Completamento: la risposta da parte del centro delle tuple di destinazione, comprensiva di tutte le informazioni sull'esecuzione dell'operazione.

La sintassi di una operazione di coordinazione op invocata su un centro delle tuple tcid è:

tcid ? op

dove tcid è il nome completo del centro delle tuple visto in precedenza. Le principali operazioni di base fornite da TuCSoN sono: out, rd, rdp, in, inp, no, nop, get e set.

Il linguaggio di comunicazione invece comprende:

- Il linguaggio delle tuple.
- Il linguaggio dei tuple template.

Dato che il medium di coordinazione TuCSoN è composto dai tuple centre ReSpecT, anche le tuple e i tuple template sono logic-based. Più precisamente, qualsiasi atomo Prolog di primo ordine è un'ammissibile tupla TuCSoN e un'ammissibile template di tupla TuCSoN.

Dopo tutte queste informazioni di base per poter comprendere quello che sarà poi il vero e proprio argomento di questa tesi, iniziamo ad avvicinarci maggiormente a ciò che ci interessa ancora più da vicino, ovvero l'insieme dei tool forniti da TuCSoN. Facciamo quindi une breve panoramica sui principali strumenti al momento a nostra disposizione:

- CLI (Command Line Interpreter)
- Inspector

Il primo è un tool a linea di comando che ci permette di interfacciarci in prima persona con i vari tuple centre. Possiamo eseguire qualsiasi operazione ci venga in mente su un centro delle tuple come se fossimo degli agenti e osservare in questo modo cosa succede da dentro. Per gli utenti alle prime armi si rivela molto utile per prendere dimestichezza con il linguaggio e capire con molta più chiarezza cosa fanno le varie primitive. Andando avanti con lo studio del sistema invece, ci viene in aiuto per osservare più da vicino

cosa succede durante l'esecuzione dei programmi, quali tuple vengono create, quali consumate e in quali circostanze. Questo è molto importante sia per controllare che tutto funzioni come ci aspettiamo, che altrettanto per capire cosa non va, come mai una tupla che ci aspettavamo fosse stata creata dal nostro agente in realtà non esiste e non permette la comunicazione e/o coordinazione tra i processi.

Per lanciare l'esecuzione del CLI usiamo il comando:

java -cp tucson.jar;2p.jar alice.tucson.service.tools.CommandLineInterpreter

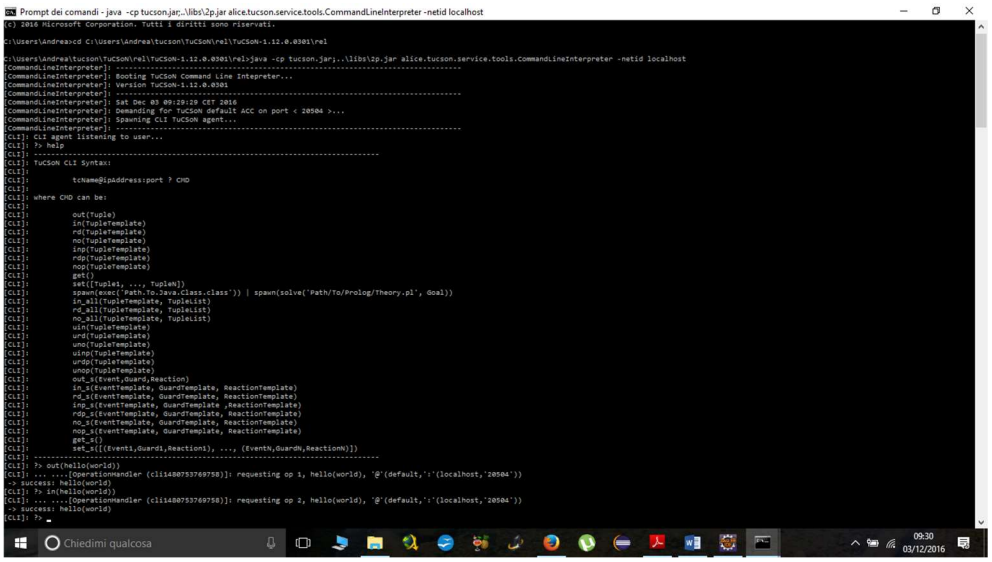

Figura 4 – Command Line Interpreter

L'Inspector invece è un tool GUI per monitorare il TuCSoN coordination space & ReSpecT VM. Per lanciare l'esecuzione usiamo il comando:

java -cp tucson.jar;2p.jar alice.tucson.introspection.tools.InspectorGUI

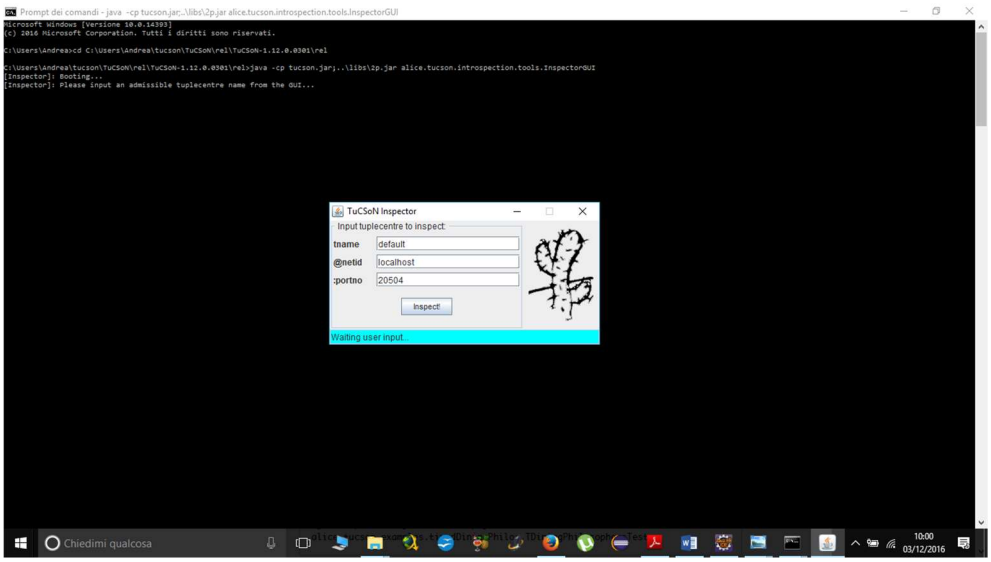

Figura 5 – Inspector

Le opzioni disponibili permettono di decidere il nome del nostro Inspector Agent, l'indirizzo IP del dispositivo che ospita il nodo TuCSoN da ispezionare, la porta sulla quale è in ascolto ed infine il nome del centro delle tuple da monitorare. All'interno del Sets tab possiamo scegliere invece cosa osservare, le possibili scelte sono:

- Tuple Space
- Specification Space
- Pending Ops: l'insieme delle operazioni in attesa di completamento
- ReSpecT Reactions: l'insieme delle reazioni innescate

L'inspector fornisce inoltre un'altra funzionalità molto utile, ovvero la StepMode che ci permette di fare il debug passo a passo delle reazioni ReSpecT.

2 Centro di Controllo TuCSoN

Veniamo ora al punto focale di questa tesi. Tutto è partito dalla volontà di estendere l'insieme degli strumenti forniti da TuCSoN con qualcosa di nuovo che ancora non esiste e possa tornare utile a chi lavora con questo sistema e a tutti coloro che ne verranno a contatto in futuro.

2.1 Analisi

Analizzando i tool già disponibili possiamo affermare che, grazie a questi, abbiamo la possibilità di lavorare sui centri delle tuple e sulle tuple stesse manipolandole e facendo esperienza con le primitive di base a noi fornite. Possiamo cioè prendere dimestichezza con il sistema basato sulle tuple, visionare tutto ciò che avviene dentro i tuple centre e soprattutto guardare l'intero sistema dal punto di vista degli agenti.

Abbiamo inoltre la capacità di analizzare le operazioni in fase di attesa, le reactions ReSpecT innescate e soprattutto poter fare tutto questo con l'utilissima funzionalità StepMode che ci permette di monitorare tutto passo per passo.

Quello che però ancora manca, e riteniamo sia utile costruire, è un'applicazione che ci permetta di analizzare tutti i tuple centres e gli agenti

attivi in determinati nodi TuCSoN già installati o comodamente installabili all'interno dell'applicazione stessa.

Ed è proprio su questo che ci concentreremo ora, ovvero sulla creazione di una nuova applicazione con GUI che ci offra le seguenti funzionalità:

- Visualizzare agenti e tuple centre già attivi e venire a conoscenza del nodo in cui risiedono.
- Installare nuovi nodi TuCSon e anche di questi tenere sotto controllo agenti e tuple centres.

2.2 Progettazione

Partiamo dal capire come vogliamo che appaia all'utente il nostro risultato finale, cioè cosa riteniamo renda chiaro e più semplice possibile l'utilizzo a chi ne farà uso.

Dopo un'attenta analisi ho deciso di impostare il tutto dividendo l'applicazione in 3 schermate:

- La prima, quella che appare all'avvio del programma, che dia la possibilità di scegliere tra le due modalità di utilizzo. La prima che, come detto, è la creazione di un nuovo nodo TuCSoN e la conseguente analisi di agenti e tuple centre che si vengono a creare, la seconda che consiste nell'analisi dei tuple centre e degli agenti attivi in nodi già precedentemente creati.

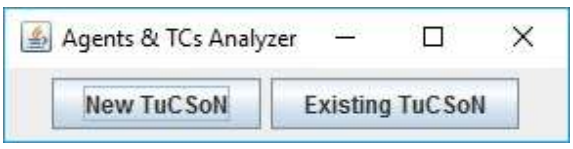

Figura 6 – Schermata di avvio

- La seconda, quella corrispondente alla selezione “New TuCSoN” che ci dia la possibilità di inserire la porta sulla quale vogliamo installare il nostro nuovo nodo TuCSoN e permetta di selezionare tramite due CheckBox cosa vogliamo venga visualizzato una volta creato il tutto. L’installazione avviene nel momento in cui clicchiamo sul pulsante “start”.

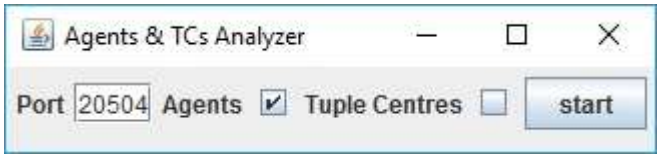

Figura 7a – Schermata “New TuCSoN”

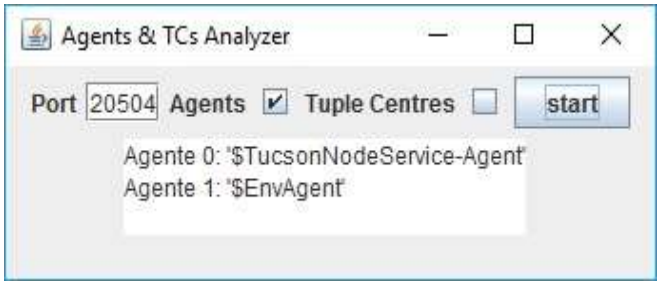

Figura 7b – Schermata “New TuCSoN” dopo aver installato il nodo tramite “start”

- La terza, rappresentante la scelta Existing TuCSoN, che ci permetta tramite due CheckBox di scegliere cosa andare a vedere dei nodi esistenti. Anche qui possiamo scegliere tra gli agenti, i centri delle tuple, oppure entrambi.

Figura 8a – Schermata “Existing TuCSoN”

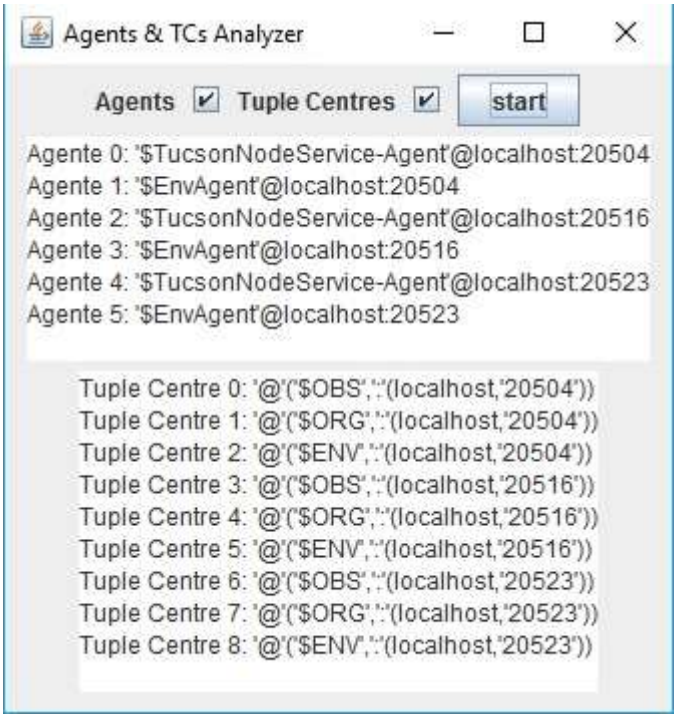

Figura 8b – Schermata "Existing TuCSoN" dopo aver avviato l'analisi

Cercando la soluzione al mio problema, è apparsa abbastanza chiara la divisione del tutto nei tre seguenti sotto problemi:

- Problema riguardante la gestione degli eventi
- Problema riguardante l'installazione di nuovi nodi
- Problema riguardante l'analisi dei nodi già attivi

2.3 Implementazione

Per quanto riguarda la gestione degli eventi ho analizzato quali componenti tra tutti quelli facenti parte della GUI avessero bisogno di un ascoltatore. Alla fine ho capito che solamente i 3 pulsanti ne necessitassero, quindi ho deciso di gestire il tutto tramite una sola classe ButtonListener a cui dal main del mio programma passo tutti gli oggetti di cui ha bisogno.

Questo Listener divide il tutto in 4 macro aree:

- Evento generato dal pulsante New TuCSoN
- Evento generato dal pulsante Existing TuCSoN
- Evento generato dal pulsante start di New TuCSoN
- Evento generato dal pulsante start di Existing TuCSoN

Nei primi due casi, il gestore non deve fare altro che nascondere la prima schermata e far apparire la seguente, corrispondente alla nostra selezione.

Negli ultimi 2 invece deve rispettivamente avviare l'installazione del nuovo nodo TuCSoN e procedere alla visualizzazione (se selezionata dall'utente) di agenti e tuple centre creati, oppure procedere alla lettura dei dati di agenti e tuple centre attivi e, sempre in base alla selezione dell'utente, mostrarli su schermo.

Passiamo ora al problema della creazione di nuovi nodi.

Dopo aver anche in questo caso analizzato con cura cosa ci si aspetta che il nostro programma esegua, ho deciso di impostare la soluzione nel seguente modo:

- Creazione di un *ArrayList<TucsonNodeService>* vuoto che ci permetterà di mantenere i riferimenti ad ognuno dei nodi da noi creati.
- Aggiunta di due metodi pubblici nel sorgente di TucsonNodeService.java ovvero:
    - *getAgents()* che ci permette di ricevere la lista di agenti attivi nel rispettivo nodo
    - *getTcs()* che ha la stessa funzionalità della precedente ma in questo caso ci restituisce i centri delle tuple
- Nel momento in cui l'utente invoca un'installazione di un nuovo nodo, questo nodo verrà creato tramite il costruttore

TucsonNodeService(int portn) e successivamente aggiunto alla nostra lista di TucsonNodeService, in modo tale da avere sempre a portata di mano il riferimento ad ognuno dei nodi che abbiamo creato

- Grazie ai metodi aggiunti precedentemente, abbiamo ora la possibilità di crearci altre due liste *ArrayList<TucsonAgentId>* e *ArrayList<RespectTC>* che ci permettono di conservare un riferimento a tutti gli agenti e i tuple centre esistenti all'interno dei nodi da noi installati e richiamarli ogni qualvolta ne avremo bisogno
- Infine, ora che abbiamo a disposizione tutto quello che ci serve, stampiamo a video ciò l'utente decide di visualizzare

Passiamo infine all'ultimo sotto problema evidenziato. Ovvero l'analisi di agenti e centri delle tuple di nodi già esistenti.

Qui il problema si fa più complicato, perché, a differenza del caso visto prima in cui creando noi i nodi ne potevamo conservare un riferimento diretto tramite una lista, ora questo non è più possibile e dobbiamo inventarci dunque una strada alternativa.

La strada più comune probabilmente sarebbe stata l'utilizzo di una socket che facesse comunicare il mio programma con i vari nodi e trasmettere di volta in volta i dati aggiornati di agenti e tuple centre.

Siccome però stiamo lavorando e studiando questo sistema basato sulle tuple, ho pensato ad una soluzione alternativa, ovvero mediante l'utilizzo di tre agenti:

- *SenderAgent*: invia le tuple dal TucsonNodeService al centro delle tuple
- *ReceiverAgent*: instanziato all'interno della nostra applicazione e che si occupa della lettura delle tuple

- DeleteAgent: richiamato dal TucsonNodeService, elimina le tuple corrispondenti ad entità non più esistenti

Facciamo però un po' di chiarezza su questi agenti e la strategia che si vuole adottare. Il principio è il seguente: ho i suddetti tre agenti, ognuno dei quali con un compito ben specifico. Il SenderAgent mi serve per andare a scrivere una tupla all'interno di un tuplecentre, arbitrariamente scelto, ogni qualvolta un agente o un centro delle tuple viene creato. Questa tupla rappresenterà quindi esattamente l'elemento appena istanziato. La tupla seguirà il seguente template a seconda di cosa vuole rappresentare:

msg(agent(Nomeagente))

msg(tc(Nometuplecentre))

Per fare ciò ho deciso di aggiungere due nuovi metodi alla classe TucsonNodeService rispettivamente di nome *writeTupleAgent*, *writeTupleTC* entrambi dichiarati protected perché potranno essere richiamati soltanto all'interno di TucsonNodeService. Questi due metodi non fanno altro che creare un nuovo SenderAgent al quale passiamo la lista degli agenti e dei centri delle tuple e richiamare il metodo *go()*; metodo go() che consiste nel creare la nostra tupla e scriverla nel tuple centre tramite la primitiva *out()*. Ultima questione riguardante il SenderAgent è stata capire in che momento andarlo a creare all'interno dell'esecuzione del servizio TuCSoN. Dopo varie prove la strategia scelta è stata posizionare un primo invio delle tuple come ultima istruzione del metodo *install()* in modo da andare a scrivere subito in blocco tutti gli agenti e i tuple centre creati all'avvio del servizio (E' qui doverosa una precisazione, tramite varie stampe intermedie, ho notato che poteva capitare che il nostro agente provasse a scrivere quando ancora il servizio non era completamente installato, quindi per ovviare a questa problematica ho adottato la soluzione di anteporre al mio agente un controllo che tutto fosse già completamente attivo

richiamando il metodo *isInstalled()*), ed un secondo invio successivo, all'interno di ogni metodo che richiamava la creazione di nuovi agenti o centri delle tuple.

Occupiamoci ora del ReceiverAgent. Questo agente entra in gioco ogni qualvolta l'utente del mio programma mi chiede di mostrare gli agenti e i tuple centre attivi. Si avvia in questo caso un agente, appunto il ReceiverAgent, che va a leggere all'interno del tuplecentre prestabilito tutte le tuple corrispondenti agli agenti e/o ai tuple centre attivi in quel momento tramite il template:

msg(agent(M))

msg(tc(M))

Utilizzando un'istruzione di tipo bulk che mi permetta in un solo colpo di leggere tutte le tuple che matchano. L'istruzione in questione è la rd_all che ci permette di leggere tuple senza consumarle, lasciandole cioè intatte all'interno dello spazio delle tuple.

Da notare una variabile di nome *rWhat* nel costruttore di SenderAgent. E' di tipo intero e può valere 0, 1 o 2. Questa ci serve perché a seconda di questo valore il mio agente capisce cosa deve andare a leggere:

- 0 legge solo gli agenti
- 1 legge solo i tuple centre
- 2 legge entrambi

Ultimo agente di nostro interesse è il DeleteAgent. Molto simile a quello appena spiegato per quanto riguarda la costruzione, ma concettualmente altrettanto diverso. A differenza del precedente questo agente viene chiamato dal servizio TuCSoN, e questo succede ogni qualvolta c'è bisogno di cancellare un agente o un tuple centre. In questi casi anche quest'ultimo non farà altro che accedere al centro delle tuple e cercare la tupla che matcha con il template desiderato. C'è però una differenza sostanziale,

ovvero la primitiva che andiamo ad usare, che in questo caso non sarà più la rd ma la *in*. Questo vuol dire che adesso noi non vogliamo più semplicemente leggere la tupla ma consumarla in modo da non renderla più accessibile in futuro, come è giusto che sia visto che stiamo eseguendo un'operazione di eliminazione agente/tuple centre. Alla stregua del caso del ReceiverAgent anche qui dovremo passare un parametro intero al costruttore del nostro agente per specificare se stiamo per cancellare una tupla di tipo agente o tuple centre.

Combinando tutte le soluzioni ai nostri sotto problemi in un'unica applicazione siamo così arrivati alla costruzione dell'applicazione che stavamo cercando.

2.4 Collaudo

Proviamo adesso ad utilizzare il nostro nuovo sistema.

Per lanciare l'esecuzione utilizziamo il comando:

java -cp tucson.jar;2p.jar alice.tucson.atcAnalyzer.ATCAnalyzer

Come possiamo aspettarci apparirà a video la schermata mostrata in figura 6, che per semplicità riportiamo di seguito:

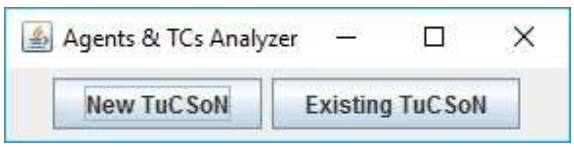

Figura 6 – Schermata di avvio

Ipotizziamo ora di voler installare un nuovo nodo TuCSoN, facciamo dunque click su New TuCSoN e ci ritroviamo anche questa volta di fronte ad una schermata a noi già familiare:

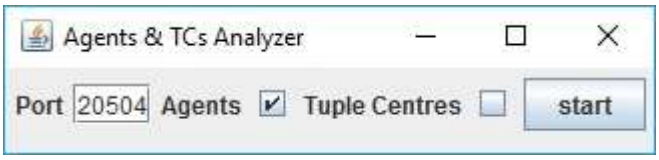

Figura 7a – Schermata "New TuCSoN"

Per semplicità utilizziamo ancora una volta la nostra cara porta di default 20504 e, spuntando sia la casella Agents che la casella Tuple Centres, quello che otteniamo è:

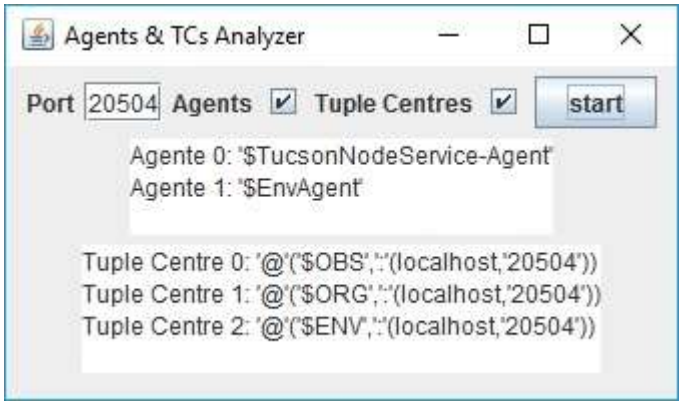

Figura 9 – Schermata "Existing TuCSoN" dopo aver avviato l'analisi

Come possiamo notare, alla creazione di un nodo TuCSoN, il sistema avvia in automatico 2 agenti:

- '$TucsonNodeService-Agent'
- '$EnvAgent'

e crea tre Tuple Centre:

- '$OBS'
- '$ORG'
- '$ENV'

Ora, per provare l'effettiva efficacia ed utilità della nostra nuova applicazione, vediamo cosa succede avviando un CLI e inserendo una tupla hello(world) nel tuple centre *default*

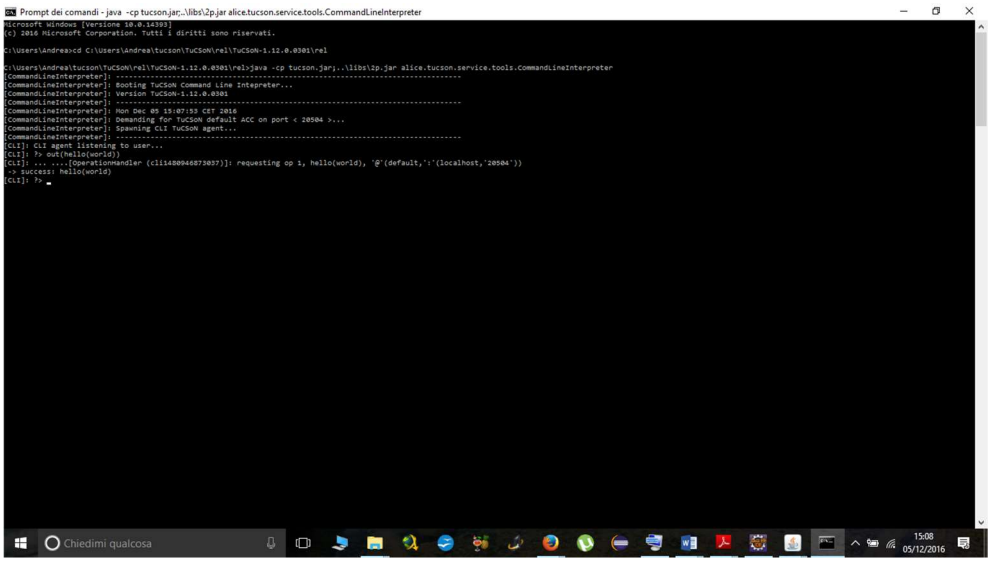

Figura 9 – Avvio del CLI e scrittura della tupla nel TC 'default'

Andiamo quindi a controllare cosa succede se chiediamo alla nostra applicazione di mostrarci agenti e tuple centre attivi.

Chiudiamo la finestra precedentemente aperta e dalla schermata principale selezioniamo Existing TuCSoN:

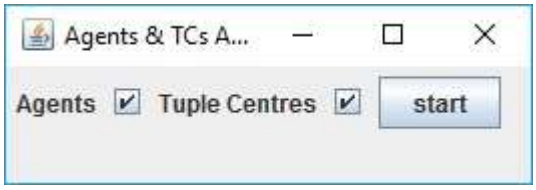

Figura 10 – Existing TuCSoN

Da qui spuntiamo entrambe le checkbox e arriviamo finalmente alla schermata desiderata:

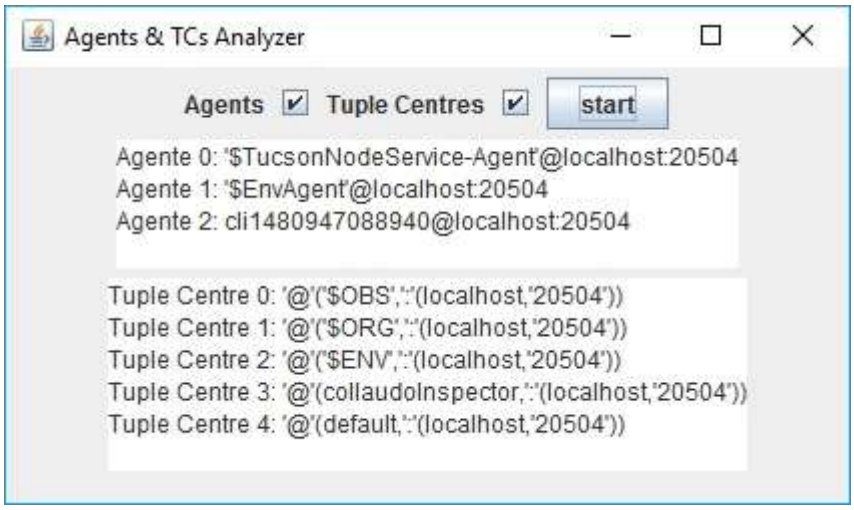

Figura 11 – Visualizzazione di agenti e TC attivi

Proprio come potevamo aspettarci, si sono aggiunti all'elenco inziale un nuovo agente 'cli1480947088940' corrispondente al CLI in esecuzione e il tuple centre default su cui siamo andati a scrivere la tupla.

In realtà guardando attentamente notiamo che si è aggiunto un'ulteriore tuple centre 'collaudoInspector' dovuto al fatto che prima di avviare il CLI, avevo fatto una prova con l'Inspector sulla TC 'collaudoInspector'. Ulteriore dimostrazione che il nostro sistema funziona a dovere.

3 Conclusioni

L'obiettivo principale di questa tesi era quello di arricchire il set di strumenti a disposizione di chi lavora con il middleware TuCSoN.

Siamo partiti studiando tutto ciò che c'era da sapere per poter analizzare con cura il problema e padroneggiare al meglio gli strumenti a nostra disposizione.

Siamo così riusciti a creare un'applicazione che ha lo scopo di arricchire quanto già fosse disponibile al servizio di progettisti e programmatori. Abbiamo superato una prima fase di analisi, progettato una strategia risolutiva adeguata e proceduto alla realizzazione pratica di quest'ultima.

In conclusione, si può affermare che il progetto è stato portato a termine con successo. L'applicazione ora a nostra disposizione vuole però essere un punto di partenza, una versione di base dalla quale partire per renderla sempre più completa ed efficiente.

Innanzi tutto sarebbe consigliabile migliorarne l'aspetto grafico, che in questa versione è decisamente basico, rendendo il tutto più pratico e user friendly. In secondo luogo si potrebbe poi pensare di ottimizzare il processo di scrittura di agenti e tuple centre per trovare una soluzione che sia più pulita di quella qui utilizzata. Infine, se se ne dovesse sentire il bisogno, aggiungere qualsiasi altra funzionalità ci si renda conto possa tornare utile durante l'utilizzo dell'applicazione.

Bibliografia

[TVS07] Tanenbaum, A. S. and van Steen, M.(2007),
Distribuited Systems. Principles and Paradigms.
Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition

[PC96] Ciancarini P. (1996)
Coordination models and language as software integrators.
ACM Computing Surveys, 28(2):300-302

[CO09] Casadei, M. and Omicini, A. (2009),
Situated tuple centres in ReSpecT.
In Shin, S. Y., Ossowski, S., Menezes, R., and Viroli, M., editors,
24th AnnualACM Symposium on Applied Computing (SAC2009),
volume III, pages 1361-1368, Honolulu, Hawai'i, USA. ACM.

[OZ99] Andrea Omicini and Franco Zambonelli.
Coordination for Internet Application development
Autonomous Agents and Multi-Agent Systems, 2(3):251-269
September 1999

[OD01] Andrea Omicini and Enrico Denti.
From tuple spaces to tuple centres.
Science of Computer Programming, 41(3):277-294, November
2001

[Llo84] JohnW. Lloyd.
Foundations of Logic Programming.
Springer, 1st edition, 1984